Printer-port data appears as bar on PC screen

Yongping Xia, EBT Inc, Torrance, CA

BBS Using the circuit in **Fig 1**, a PC's printer port can accept 8-bit parallel data. The 74HC241 in the circuit is a data buffer as well as a high/low nibble selector. The Borland C program in **Listing 1** reads the high and low nibbles, reforming the 8-bit data and converting them to a vertical bar on the PC's screen. Moreover, the program can compare the input data with preset high and low limits, sending the results of this comparison back out through the printer port. You can obtain a copy of the **listing** from **EDN BBS/DI_SIG #1432.** Because the printer port powers the IC, the circuit requires no external power. (DI #1432) EDN

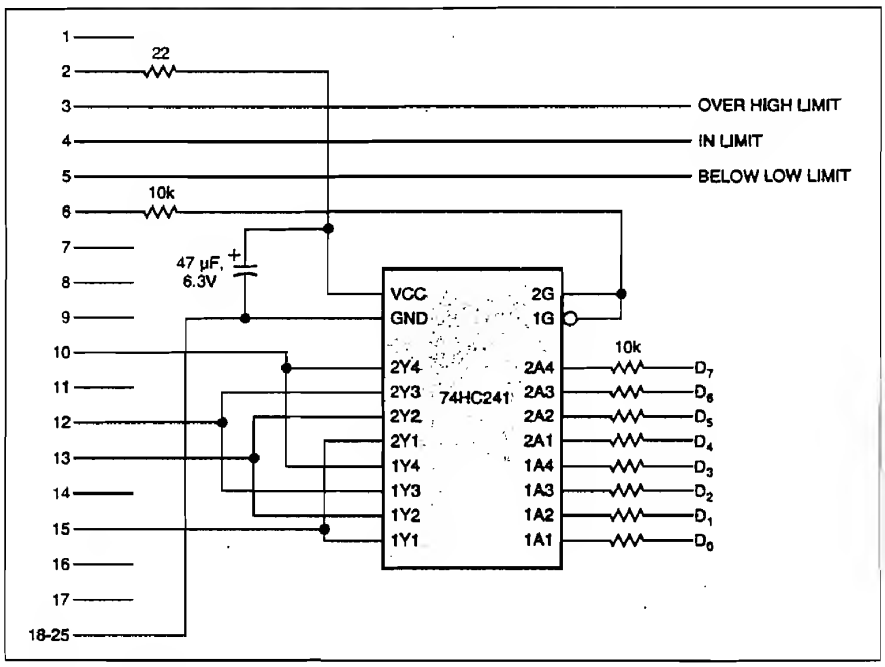

Fig 1—This simple buffer breaks up 8-bit data into a pair of nibbles.

To Vote For This Design, Circle No. 347

Listing—Command-and-display program for printer-port data buffer

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <bios.h>

#define POWER_ON      0x01
#define LOW_4BIT      0xaf
#define HIGH_4BIT     0x10
#define CLEAN_OUT     0xf1
#define OVER_LIMIT    0x02
#define IN_LIMIT      0x04
#define BELOW_LIMIT   0x08
#define HIGH          0x90                  /* high limit                */
#define LOW           0x70                  /* low limit                 */

typedef unsigned int WORD;

int  data, out_port, in_port, out=0;
char msg[80];

main()
{
  find_port();
  init_graph();
  init_screan();
  do {
       out &= LOW_4BIT;
       outportb(out_port, out);             /* set port to read low nibble  */
       data=(inport(in_port)/8)&0x0f;       /* read low nibble              */
       out |= HIGH_4BIT;
       outportb(out_port, out);             /* set port to read high nibble */
       data+=(inport(in_port)*2)&0xf0;      /* combine high and low nibblee */
       out &= CLEAN_OUT;
       outportb(out_port, out);             /* clean compareson output      */
       if (data>HIGH)
  {
    out|=OVER_LIMIT;
    outportb(out_port, out);                /* send out over-limit output   */
  }
       else if (data<LOW)
  {
    out |= BELOW_LIMIT;
    outportb(out_port, out);                /* send out below-limit output  */
  }
       else
  {
    out |= IN_LIMIT;
    outportb(out_port, out);                /* send out in-limit output     */
  }
       setviewport(0, 0, 639, 479, 1);
       setfillstyle(1, RED);
       bar(70, 10, 85, 286-HIGH);           /* display over-limit bar       */
       setfillstyle(1, GREEN);
       bar(70, 286-HIGH, 85, 286-LOW);      /* display in-limit bar         */
       setfillstyle(1, RED);
       bar(70, 286-LOW, 85, 286);           /* display below-limit bar      */
       line(70, 286-data, 85, 286-data);
       line(70, 285-data, 85, 285-data);    /* display data mark            */
       setviewport(68, 320, 95, 340, 1);
       clearviewport();
       sprintf(msg, "%d",data);
       outtext(msg);                        /* show data on screen          */
       delay(100);
     } while (!kbhit());                    /* quit if any key hit          */
  clean_up();
  return 0;
}

init_screen()                               /* initialize display screen    */
{
  setbkcolor(BLUE);
  setcolor(WHITE);
  line(2,2,637,2);
  line(5,4,634,4);
  line(2,2,2,477);
  line(5,4,5,475);
  line(5,475,634,475);
  line(2,477,637,477);
  line(634,5,634,475);
  line(637,2,637,477);
  line(5,350,634,350);
  setviewport(450, 440, 630, 460, 1);
  sprintf(msg, "press any key to quit");
  outtext(msg);
  setviewport(0, 0, 639, 479, 1);
  return 0;
}

init_graph()                                /* initialize graphic mode      */
{
   int gdriver = DETECT, gmode, errorcode;
   initgraph(&gdriver, &gmode, "");
   errorcode = graphresult();
   if (errorcode != grOk)
   {
     printf("Grephics error: %s\n", grapherrormsg(errorcode));
     printf("Press any key to halt:");
     getch();
     exit(1);
   }
   return 0;
}

find_port()                                 /* find printer port's address  */
{
   out_port=*(WORD far *)MK_FP(0x0040,8);
   in_port=out_port+1;
   out|=POWER_ON;
   outportb(out_port, out);                 /* power on                     */
   delay(1000);
   return 0;
}

clean_up()                                  /* close graphic mode           */
{
   closegraph();
   return 0;
}
```

PC printer port controls frequency divider

Bogdan Manolescu, Microelectronica, Bucharest, Romania

BBS

In **Fig 1**'s circuit, two cascaded synchronous presetable binary counters, IC_1 and IC_2, can derive signals having a frequency of f_{CLK}/N. In the circuit, a simple oscillator generates f_{CLK}; however, you can substitute any triggerable source.

An IBM PC supplies the binary-coded integer divisor N (N=255 max) via eight pins of its printer port. Two additional control lines (pins 1 and 14 of the printer port) provide start and reset functions. The signal that starts the oscillator (COM=0) also enables the first counter, IC_1.

The counters, wired to count down, activate the overflow output of IC_2 when the counters reach zero. The overflow signal then enables the counters' parallel loading of the integer divisor N. The Turbo C++ program in **Listing 1** controls the frequency dividers' operation.
(DI #1431) EDN

To Vote For This Design, Circle No. 346

Listing 1—Frequency-divider control program

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <ctype.h>

#define     OUT_PORT  0x378  /* printer output  port address */
#define     CTRL_PORT 0x37A  /* printer control port address */

int main(void)
{
  int n; char c;
  for (;;)  {
    clrscr();
    printf("Input the divisor (between 1 and 255): ");
    scanf("%d", &n);
    printf("\nStart ? (y/n)");
    if(tolower(getch()=='y')) {
      outportb(OUT_PORT,n);       //send out divisor
      outportb(CTRL_PORT,0x02);  //start oscillator and enable counter1
      printf("\n\n\tStop with any key...\
        \n\tESC to exit...");
      c=getch();
      if(c==0x1b) break;
      outportb(CTRL_PORT,0x01);  //reset the counters
      delay(1);
      outportb(CTRL_PORT,0x03);
      delay(1);
    }
  }
  return 0;
}
```

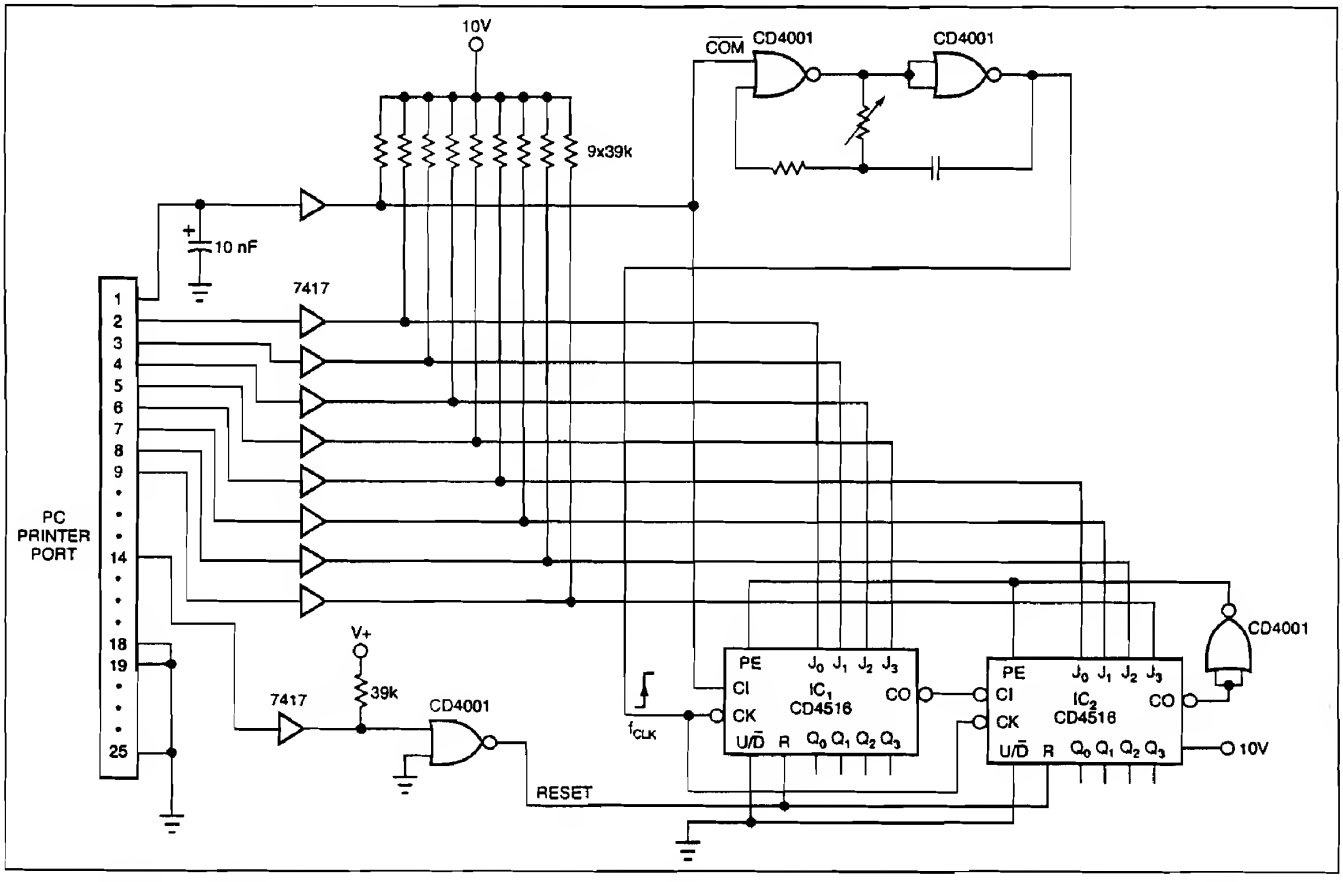

Fig 1—An IBM PC supplies a binary-coded integer divisor circuit to this circuit's pair of synchronous, presettable binary counters.